ARTICLE IN PRESS

Available online at www.sciencedirect.com
SCIENCE DIRECT®

ELSEVIER

Information & Management 2033 (2002) 1–14

INFORMATION & MANAGEMENT

www.elsevier.com/locate/dsw

Quality management metrics for software development

John S. Osmundson[a,1], James B. Michael[b,*], Martin J. Machniak[c,2], Mary Alice Grossman[d,3]

[a]Department of Information Sciences, Naval Postgraduate School, CC/Os, 833 Dyer Road, Monterey, CA 93943, USA
[b]Department of Computer Science, Naval Postgraduate School, CS/Mj, 833 Dyer Road, Monterey, CA 93943, USA
[c]Space and Naval Warfare Systems Center, OTC-2, Code 2334, 53560 Hull Street, San Diego, CA 92152, USA
[d]NASA Dryden Flight Research Center, P.O. Box 273, MS 4840A, Edwards, CA 93523, USA

Received 22 July 2001; received in revised form 15 April 2002; accepted 23 August 2002

Abstract

It can be argued that the quality of software management has an effect on the degree of success or failure of a software development program. We have developed a metric for measuring the quality of software management along four dimensions: requirements management, estimation/planning management, people management, and risk management. The quality management metric (QMM) for a software development program manager is a composite score obtained using a questionnaire administered to both the program manager and a sample of his or her peers. The QMM is intended to both characterize the quality of software management and serve as a template for improving software management performance. We administered the questionnaire to measure the performance of managers responsible for large software development programs within the US Department of Defense (DOD). Informal verification and validation of the metric compared the QMM score to an overall program-success score for the entire program; this resulted in a positive correlation.

Keywords: Metrics; Software management; Software process

1. Introduction

Quality of management has long been a concern within the software engineering community. The term ''software crisis'' was coined in the 1960s to refer to problems in developing software on time, within budget, and with the properties that the software was usable and actually used. The General Accounting Office reported in 1979 [23] that of the government software development projects studied:

- more than 50% had cost overruns;
- more than 60% had schedule overruns;
- more than 45% of the delivered software could not be used;
- more than 29% of the software contracted for was never delivered;
- more than 19% of the delivered software had to be reworked.

* Corresponding author. Tel.: +1-831-656-2655; fax: +1-831-656-2814.
E-mail addresses: josmundson@nps.navy.mil (J.S. Osmundson), bmichael@nps.navy.mil (J.B. Michael), machniak@spawar.navy.mil (M.J. Machniak), mary.grossman@mail.dfrc.nasa.gov (M.A. Grossman).
[1] Tel.: +1-831-656-3775; fax: +1-831-656-3679.
[2] Tel.: +1-619-524-3473; fax: +1-619-524-3507.
[3] Tel.: +1-661-276-5531; fax: +1-661-276-2792.

0378-7206/02/$ – see front matter © 2002 Published by Elsevier Science B.V.
PII: S0378-7206(02)00114-3

Since that report was released, the software engineering community has attempted to improve the software development process. For example, the well-known capability maturity model (CMM) identifies key practices required to improve organizations' software development processes. The CMM was codified into five levels of increasing organizational maturity levels. More recently a personal software process (PSP) has been introduced, defining key practices required to improve an individual's software development processes [11]. There is evidence that attaining high levels of capability as defined by the CMM has resulted in improvements in the management of the development of software-based systems [3] and further evidence has shown the potential benefits to be derived by enacting a PSP [7].

The quality of software development management tools has improved over the past 30 years. However, many of the same challenges, such as keeping software development projects on schedule and within budget, remain today. The Standish Group report [9] in 1995 found that, on average, approximately 16% of software projects were completed on time and within budget. In large companies the record was even worse: only 9% of the projects were completed on time and within budget. Moreover, the projects that were completed contained only approximately 42% of the originally proposed features and functions. Also, US-based companies and government agencies spent $ 81 billion for canceled software projects and these same organizations paid an additional $ 59 billion for software projects that were completed, but exceeded their original time estimates. According to Fabian-Isaacs and Robinson [5], ''software development projects are notorious for running over budget and behind schedule''. The Center for Project Management in San Ramon, CA reported that 99% of commercial software products are not completed on time, within budget, or according to specifications, and that the average project is underestimated by 285%.

CMM level-four compliance requires the development organization to collect metrics that measure the effectiveness of the development process, while CMM level five requires that the organization use the metrics continuously to improve its development process. IEEE Standard 12207.0 and the earlier MIL-STD-498 include metrics that might be gathered during the software development process. Examples include software size and complexity, software units developed over time, milestone performance, and problem/change report status. Metrics are defined for the software development process and the software product but not for the quality of the project management. One can argue that in order to systematically go about improving the management of software projects, it is necessary to measure the quality of project management. Program-management tools have been developed to assist the program manager in estimating the cost and schedule of software programs. However, the estimation tools available assume consistent and high-quality program management.

One of the earliest and most widely used software project cost-estimation models is COCOMO [2]. The basic, intermediate, and detailed COCOMO models are based on the results of analyzing 63 software projects and applying regression analysis in order to predict software development cost as a function of software size and other factors. The intermediate and detailed COCOMO models take into account attributes of the software product, of the computer hardware, development personnel, and the project. Examples of project attributes include the use of software tools and the required development schedule. Intermediate cost estimates are based on the estimated number of lines of code (LOC) to be developed and then these estimates are adjusted by applying multipliers determined by rating the project with respect to the attributes. For example, a project completed under an accelerated schedule is estimated to cost more. However, COCOMO does not take into account the quality of project management.

Boehm wrote, ''poor management can increase software costs more rapidly than any other factor'' and ''despite this cost variation COCOMO does not include a factor for management's quality, but instead provides estimates which assume that the project will be well managed''. Among the reasons Boehm gave for not including management quality was that management quality ratings are not easy to determine.

If the quality of the software program management were measurable and available as input to costing and scheduling tools, the resulting estimates could pinpoint areas of software program management in which improvement needs to be made. Being able to measure the quality of management of software projects objectively allows development of more accurate cost models and would also provide a means for improving

software project management through assessment, feedback, and correction. In this paper, we introduce such a metric that is repeatable, termed the *quality management metric* (QMM) [16]. We also discuss the informal and more formal validation of the metric.

The QMM is computed from the quantitative answers to a structured set of inquiries, in a questionnaire consisting of two parts:

(1) a set of paired choices between statements that reflected possible management actions on a software program, and
(2) a set of questions requiring a yes, no, or not applicable answer.

The questionnaire was designed to eliminate essay-type answers and to minimize, as much as possible, subjective assessments.

The questionnaire addresses four areas of software management considered to be the most important: requirements management, people management, risk management, and planning/estimation management. We assume that, collectively, measures in the four areas can give an objective view of the quality of software management for a specific software development program. Thus, two programs scoring equally on product and process metrics can be further measured and compared on the basis of the quality of their management, thereby providing a more comprehensive look at a software program.

2. Requirements management

The management of requirements is an important measure of the quality of program management. Constraints can be in the form of mandates to employ a certain development process, a selected architecture, or by a predetermined set of requirements.

The program manager must identify and ensure that all major stakeholders are involved in the initial elicitation and articulation of software requirements. Failure to include all parties at the start will most likely spell trouble down the line [25]. McConnell [21] refers to the product specification as the software program's ''compass'':

> ...without one, you can perform the work of Hercules and still not produce a working product because the work in aggregation hasn't been aimed in any particular direction. Without good direction, any individual's work can go the wrong direction and different people can work at cross-purposes.

Program managers can regard requirements as the contract between the developer and the customer on a program, and manage the customer's expectations by managing the requirements [14].

Requirements management focuses on managing the process of extracting, developing, defining, and refining the requirements of a software program [1]. Product and process metrics do exist [10]. Davis and Leffingwell [4] state that requirements are capabilities and objectives to which software must conform and are the common thread for all development (and maintenance) activities. Requirements management is the process of eliciting, documenting, organizing, and tracking changing requirements and communicating this information across the project team. Implementing (quality) requirements management ensures that iterative and unanticipated changes are maintained throughout the project lifecycle.

Quality management of a program's requirements must establish procedures and structure to ensure that requirements specifications are complete, consistent, readable, unambiguous, traceable to their origin, and do not arbitrarily contain design stipulation. Each requirement should be a singular idea. Good management addresses the requirement attributes, including the following: managing customer benefit, the requirements author and/or responsible parties, the corresponding effort, the development priority, rationale, and relationships to other requirements. The effort in tracking status, dates, and versions also is a determinate of quality management.

A quality program manager will, among other things, facilitate the user/customer needs into requirements that can be implemented. This process happens in one of the two ways. The first is the direct procedure. Users convey in any number of ways their needs to program management, which in turn develops the formal requirements to which the developers code. In the indirect procedure, the users convey their needs directly to the development team, which in turn develops prototypes that the users can validate prior to detailed design, coding, and testing. Program manage-

ment adjudicates between user and developer during the indirect process and assists in the specification of formal requirements. However, the formal requirements serve mainly as a record of what has been performed [15].

Although these tools enable one to build lists of requirements, it is the responsibility of the program manager to establish requirements prioritization [26]; this is typically derived from tradeoff analyses that take into consideration the level of availability of various resources, such as the development experience of the software development team, project budget, time allotted to complete the project, and availability of software development tools and reusable components. Identifying all the requirements upfront and then developing the product is idealistic in today's software environment. Requirements change for many reasons [8]. It is the program manager's responsibility to establish controls such as a change-and-configuration management (CCM) process. CCM helps direct and coordinate those changes so they can enhance, rather than hinder software development. The CCM procedures must be easy to understand and consistent. It is well documented that time and cost increase almost exponentially when requirements are changed late in the development process. The program manager must choose to ''freeze'' requirements at some point, but establish the framework for a follow-on version release or block any upgrade: unlike most durable goods, software systems are ever changing.

3. Estimation/planning management

Estimations are the basis from which planning is performed on a program [13]. Planning a software product development requires a frame of reference and an ability to measure against it. The program manager has three major measures with which to estimate the program: products, processes, and resources [22].

Product measures generally refer to volume, such as LOC. The measure can be the whole product or various elements, such as modules, components, or manuals. Measurement is accomplished by phase, such as the amount of code produced in the implementation phase or the LOC changed during unit testing. Measures of other product attributes might include system throughput, cyclomatic complexity, module coupling, and function points (FP). Process measures quantify behavior, strategies, and execution of the process used to develop the product. One general category of process measures is event counts, such as the number of defects found in test, requirement changes, or milestones met. Another general category concerns time measures, such as cycle time: time to complete a project. In highly competitive markets, cycle time, or deployment, may be more important than reducing development costs.

Resource measures refer specifically to labor hours required for product development. Monetary cost typically becomes an estimated outcome from process, product, and resource measures. Estimation utilizing all three measures can be used for planning schedules and costs. Subsequent tracking of metrics throughout the program will aid program updates and provide a basis for planning future programs. For example, program management can use work breakdown structures as a tool to identify and track important tasks, milestones, and deliverables throughout the program and life cycle of a software-based system. Once initial costs and schedules are derived from estimations, progress tracking and schedule-and-cost adjusting become key factors in the success of the software development program.

Establishing and tracking earned value is recommended as a way to measure program progress. By assigning value to a developer's work package, its current cumulative value can be compared to the estimated and actual cost to complete to give a more accurate measure of schedule-and-cost progress. The program manager must set up a structure to use product, process, and resource measures in a software program, and it is the program manager's responsibility to ensure that the measure being used will yield the most accurate and useful results.

4. People management

If one person could perform all the software development tasks, there would be no need for the management of people. How management recruits, organizes, and treats human resources is instrumental to the success or failure of any endeavor. Software development is an intellectual activity that requires creative

problem solving before and during the application of software processes, methodologies, and tools. People management encompasses not only such issues as the program manager's ability to allocate human resources and ensure an appropriate work environment, but it also requires communication and leadership, including the structure for communication and mentoring for the entire program. The QMM is intended to examine questions such as: does the management create the proper environment through good working conditions and an appropriate reward structure?

5. Risk management

An overarching theme that runs through each of these sections is risk management. Ultimately, it is the management's ability to identify and manage high-risk elements early in the process; this will have an impact on the success or failure of a software program [24]. We define risk exposure as the product of the probability of an unwanted event and the loss experienced if the event occurs. Such problems might have an adverse impact on the cost, schedule, or technical success of the program; the quality of products; or team morale. Because non-trivial software development programs typically do not run as planned, every software program carries with it some degree of risk [12]. Therefore, requirements, estimation/planning, and people management all engender some level of risk. Risk management is the process of identifying, addressing, and mitigating the effects of unwanted events. It is critical in measuring the management quality of a software program.

The cost of managing risk is relatively low at the start, but increases as the program progresses. The factor takes into account any structure that promotes success in the software development environment by considering individual risks, assessing individual impact, determining a probability of occurrence, and planning a mitigation strategy. Program management's judgments within the established structures will vary, and can ultimately determine the success or failure of a risk-management effort. However, the establishment of structure dedicated to these practices can be objectively measured and provide an indication of the quality of program management.

6. The QMM questionnaire

The approach used to develop the QMM included searching the literature, interviewing senior program managers, and conducting focus group meetings.

Focus groups (generally with 4–12 individuals per session) consisted of a wide range of government and private industry software professionals each involved with or previously involved with US Department of Defense (DOD) software development projects. Individual experience was from 2 to over 20 years. Software categories ranged from program managers to programmers. The predominant software language experience was in Ada, C, and C++. However, many participants had experience with other languages and other software projects outside DOD. Sessions were conducted with facilitators, structured to maintain focus on the issues, and with care to avoid bias of the outcomes.

The QMM measures the quality of management for and in a specific software program. The overall goal was to develop an objective, standardized metric to which program management could be compared and ranked, thus providing a baseline for quantifying improvement. This metric compares the same management on different software programs or at different times during the same program. Metric development is difficult, because the quality of management may be very subjective. Words that prompt subjective responses, such as "feel", "think", and "believe", were avoided as much as possible in the QMM questionnaire. Answers were constrained to enable scoring to a scale.

Part one of the questionnaire contains pair choice questions. The person filling out the questionnaire must choose one of the two statements that best described their program. The choice did not have to match exactly; it should just be the closest. Each pair statement represented two differing ideas in order to ascertain a tendency of the individual. Often the pair choices were repeated with different wording to confirm earlier choices and measure the strength of the tendency. The survey format, with the proper mix of questions and repetitions, was intended to be used for reaching consensus on issues and to measure the strength of tendencies. Each section had a maximum score of 70 points. The risk, estimation/planning, and people management sections had 70 questions each.

The requirements management section had 50 questions and included an alternate block of 16 questions depending on the software development strategy used.

Part two of each questionnaire consisted of yes–no–n/a (n/a: not available) questions. Instead of asking open-ended questions that participants could answer in a variety of ways in essay form, this format standardizes the responses for easier comparison. It is user-friendly for conducting surveys, requiring minimum writing by the participant. Each yes, no, or n/a choice has an associated point value, based on the relative importance of the question. The use of the ''n/a'' box was discouraged. However, it was used in cases in which the program manager did not have direct control over the issue that was raised in the question. For example, a government program manager may not have direct hire/fire authority over development personnel. Thus, if the survey question asked whether the program manager has direct hire/fire authority over the personnel, the appropriate answer would be n/a, because it allowed the interviewee to indicate that the program manager is constrained and thus does not penalize the program manager for factors beyond his or her control.

Each section had a maximum value of 62 points. The estimation/planning, people, and risk-management sections had 50 questions each. The requirement section had 47, including an alternative block of 6 questions, depending on the development strategy used. The complete survey, including both parts for all four sections, contained 457 questions.

The choice of this questionnaire format sought to dissect complex decisions into their basic components of choice. Objectively evaluating and comparing overarching program structures and policies required a survey that identified and evaluated the basic level of decisions in all relevant aspects of software management. To avoid any pre-bias tendency of one response over another, administration of the questionnaire was conducted so that the subject was unaware of the point value of each response.

The questionnaire for the management of requirements evaluated the program manager on establishment of procedures. These questions did not seek to determine the quality of judgments on any specific decision. The thrust of the questions was to establish the structure, if any, laid out by the program manager in the area of requirements. Examples of requirements pair choice selection questions were the following, where the interview subject was asked to mark the box of the most appropriate answer:

Formal requirements list	Informal requirements list
Requirements taken as is from customer	Look to reformulate, interview in-depth, or otherwise re-validate

The estimation/planning management section did not seek to choose or require a specific estimation technique. This area sought to quantify the management effort of the estimation process. The questions address whether the choice of an estimation technique was appropriate and how well that technique was implemented. Examples of estimation/planning management pair choice selection and yes–no–n/a questions are:

Estimates by algorithmic methods	Estimates by analogy
Management only on estimations	All team members involved in estimation process

	Yes	No	n/a
Code reviews planned in schedule			
Work breakdown structure developed			

Because people management encompasses many distinct areas, each of which was highly weighted in importance, the questionnaire was divided into four subsections: human resources, leadership, communication, and technical competency. The leadership questions reflected the personal leadership skills exhibited and the leadership mentoring provided by the program manager. The communication questions sought to ascertain the communication protocols set up for the program organization and used individually by the program manager.

Examples of people management pair choice selection and yes–no–n/a questions are:

Keep people well informed	Only as much knowledge as necessary for their work
Coders notebook, weekly accomplishment reports required	Not required

	Yes	No	n/a
PM is accessible in person by each team member			
PM attempts to motivate individuals on the program team			

The questionnaire also was used to ascertain the structures used by program management for identification, monitoring, and managing risk. The questions determined whether the program manager had set in place strategies and personnel to implement risk assessment, explore, and prioritize all reasonable risks. Does the program manager have an active risk-management program and established procedures to monitor the risks and update the plan? The goal was to ensure that the program manager had, for each identified risk, an integrated mitigation strategy. Examples of risk-management pair choice selection and yes–no–n/a questions are:

Risk management is formal and documented	Risk management is informal, if at all
Risk status tracked	Not tracked

	Yes	No	n/a
Risk management is formal and documented			
Risks are tracked			

It is difficult to measure individual judgments about risk management. What can be measured is whether the program manager has performed risk-management elements.

7. Methodology and scoring

The methodology is illustrated in Fig. 1. The QMM survey instrument was administered to selected program managers and software developers. Raw QMM scores were weighted, converted to a 1–10 scale, and then compared to subjective success scores estimated for the same programs by the same survey subjects.

The point totals from each of the two questionnaire parts per section were entered on the QMM Summary Score Sheet. Point totals for part one and part two were then added together to determine the total points for each section. These were multiplied by their relative Importance Coefficient (IC) to yield a weighted score. After weighted scores were determined for each of the four sections, they were summed together to yield the QMM score.

The IC was determined from the relative rankings of importance of each of the sections. Experienced software professionals provided the data to determine the IC through the focus groups [17,19] and one-on-one

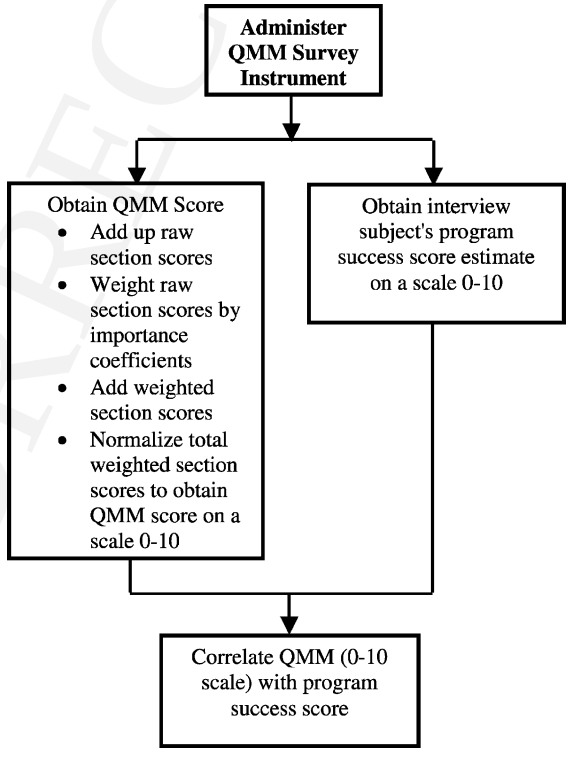

Fig. 1. QMM survey instrument methodology.

interviews [18,20] only after thorough explanation and understanding of each category.

The QMM equation is given in Eq. (1):

$$QMM = 0.92RqM + 0.67EPM + 0.55RkM + 1.86PM \quad (1)$$

where RqM is the requirements management metric score, EPM the estimation/planning metric score, RkM the risk-management metric score, and PM is the people management metric score.

The QMM ranges from 528 points to −130.9, as part two contained negative point response values. The QMM percentage score is a derived measure of the QMM score. To obtain a QMM score scaled from 1 to 10, the survey minimum possible score was normalized to 0: 130.9 was added to the survey minimum possible score. Correspondingly, this was also added to both the survey maximum QMM score and to the actual QMM score obtained. The normalized QMM score obtained from the survey was then divided by the normalized survey maximum possible QMM score of 659 and then multiplied by 10.

Once the survey was completed, the interviewees were asked to rate the success of the program at the point in time when the program was being evaluated, using a scale of 0–10. To assist the interviewees in visualizing the scale, the interviewer drew and labeled a scale (as shown in Fig. 2) and the interviewees were asked to place an "X" at the place that represented the success of the program at this point in time. Next the interviewer asked the interviewee for a numerical value to associate with the "X" on the scale. Zero was defined as abject program failure with no worthwhile product. Ten was defined as an absolutely perfect software product with flawless program execution.

The survey participant's QMM score was compared to his or her individual overall success score and to the mean overall success score of the program. The goal was to determine any correlation between the participants' QMM score, their individual success ranking of the overall program, and the mean success ranking of the overall program.

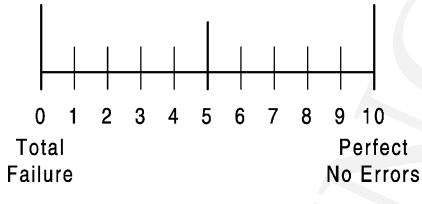

Fig. 2. Overall program score scale.

8. Informal QMM validation

Three software programs were evaluated during 1999 at the US Space and Naval Warfare (SPAWAR) Systems Center. The program manager and one program development team member evaluated program A, the program manager and two program development team members evaluated program B, and the program manager and one program team member evaluated program C.

Table 1 summarizes the resultant scores of the three programs. The subscript "PM" indicates the program manager's survey results and the number in subscript indicates a participant's survey results other than the program manager. The mean success score of a program includes the individual success ranking scores by the individuals participating in the survey and others associated with the program in some way in which they can judge the success of the program.

Although there is a strong overall correlation between QMM scores to success scores, as shown in Table 2 there is a negative correlation between the program managers' assessments and those of the developers. There are strong correlations for program manager QMM scores to program manager success scores and a similar strong correlation for corresponding developer scores, but weak or negative correlations between program manager and developer assessments; this was due to the effect of the assessment of the manager of program B within the small sample set.

The summary sheets for program A revealed a weak risk-management section, but overall the program was highly structured and enjoyed good technical success with its deliverables. Program C was a smaller program that was relatively unstructured, with essentially no risk management, little planning and poor requirement extraction. However, the program delivered a usable product, due to strong practices in the people management portion and a technology that was relatively straightforward. Program B exhibits a significant divergence from the scores of the program

Table 1
Results of informal QMM validation

	Program A		Program B			Program C	
	Participant A_{PM}	Participant A_1	Participant B_{PM}	Participant B_1	Participant B_2	Participant C_{PM}	Participant C_1
QMM score	338	322	386	106	47	198	189
QMM score (0–10)	7.12	6.88	7.85	3.59	2.70	4.99	4.86
Success score (0–10)	7	7	9	4	3	4	4
Mean success score	7		4			4	

Table 2
Correlations of data from informal validation

	Score		
	Dev. QMM	PM success	Dev. success
PM QMM score	−0.20	0.99	0.14
Dev. QMM score		−0.35	0.94
PM success score			−0.02

manager and the other team members. This program appeared to have a dichotomy in perception, and further interviews with others in the program indicated that there were significant management issues that needed to be resolved. The overall conclusion of the informal validation was that the QMM had promise of being a valid approach to developing a quality management metric, and that further study was warranted.

9. Extended QMM validation

The QMM survey instrument was subsequently used by Grossman [6] to extend the informal validation by measuring the performance of 10 program managers on US DOD software development programs. These 10 were asked to complete the QMM questionnaire and, in addition, the survey instrument was given to one to two members of the development team who were knowledgeable about the overall practices and success of each of the programs. A requirement for choosing the individual development team members was that they had a good understanding of the overall program and were knowledgeable about the management practices and infrastructure implemented by the program manager throughout the program. We tried to choose individuals whose experience was not limited to specific areas of the program.

The program manager was asked to determine a specific point in time on the program, such as a milestone or delivery, for the evaluation of the program management and to define it so that the individual development team members would be able to identify the time that was selected in order to be able to evaluate the program for that same point. The survey instrument was applied to the interviewees in one of two ways: one-on-one personal interview that lasted approximately 2 h or via an electronic copy distributed and returned by electronic mail. In 8 of the 10 programs (17 surveys in total) the survey instrument was applied in personal interviews; in the remaining two, H and J (four surveys total), the survey instrument was applied through electronic mail. The interviewer also asked the interviewee for feedback on the survey instrument itself. For example, they were asked:

- Were there any questions that did not make sense?
- Were there words that the interviewee did not understand?
- Was the survey instrument too long?
- How would they improve the survey instrument?
- What was their overall impression of the survey instrument?

This feedback was collected to determine the level of frustration that the survey induced in the interviewee and to provide glimpses into the viability of the survey instrument for future improvements. In order to encourage complete and open participation in the QMM survey, the interviewees were assured at the beginning of the survey process that the results would be reported anonymously. To this end, the program data reported here is reported as program A, B, C, etc. The minimum time required for completing the survey

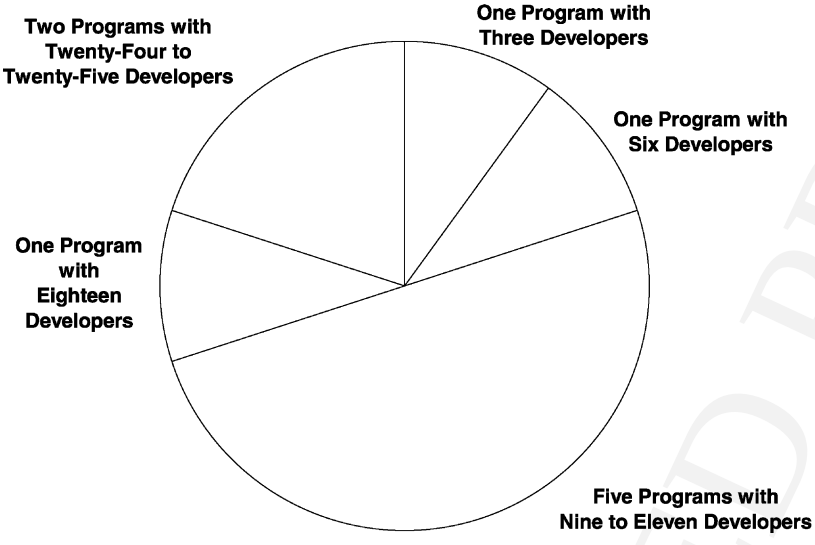

Fig. 3. Size of programs surveyed.

instrument in a personal interview was 1 h; the average time required to complete the survey instrument was 2 h and the longest time was 4 h.

Figs. 3 and 4 provide information about the characteristics of the US DOD projects included in the extended QMM validation.

Results of program managers' assessments are summarized in Table 3. The first column gives the program-identifying letter, the second column gives each program manager's subjective score on a 0–10 scale, and the third column gives the program manager's total QMM score, translated to a 0–10 scale. Table 4 gives the equivalent set of scores for the individual developers (IND).

The data was examined to determine if there were any obvious trends. The possibility of the program

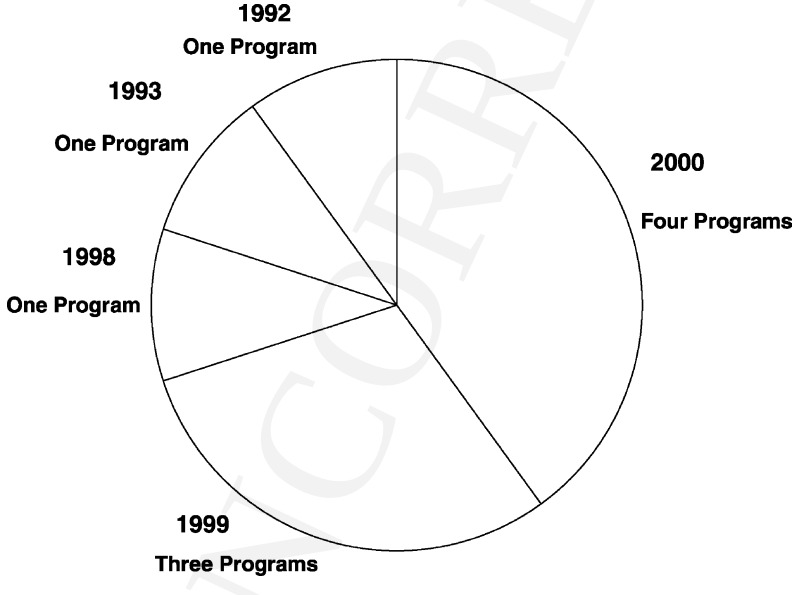

Fig. 4. Time frame of programs surveyed.

Table 3
Program manager (PM) QMM scores from extended QMM validation

Program	PM score					
	Program	QMM	Req. Mgt.	Est. Plan.	People Mgt.	Risk Mgt.
A	9.9	6.8	5.2	6.2	8.6	4.0
B	7.5	8.9	8.6	9.3	8.8	9.5
C	8	8.1	8.8	7.5	8.8	5.1
D	7	7.0	7.2	5.0	7.9	5.9
E	4	5.2	5.2	4.5	6.1	2.6
F	9	5.4	5.5	3.7	7.3	0.7
G	7	6.8	6.0	6.6	7.8	4.9
H	6	7.7	7.1	7.5	7.8	8.3
I	3	4.8	4.8	3.8	6.3	0.9
J	7	8.2	8.6	6.9	8.4	8.1

managers consistently rating the success of the program higher than the corresponding QMM percentage score was not found in the data. In one instance, program A, the program manager rated the program overall far higher than the QMM percentage score, the individual development team member overall program score, and the QMM percentage score. The survey results obtained from program F were contrary to our expectations. It did not utilize many of the currently accepted software engineering management processes and procedures, such as formal risk management. Thus, we expected the ranking of the success of this program to be low. On the contrary, the program was ranked as highly successful by the software developers and the customers with an overall program-success score of 9.0, yet the QMM percentage score consistently ranked the program as a 5.4. The exciting part of the discovery is that it appears that the QMM survey instrument does measure the successfulness of the implementation of currently accepted software engineering management practices as defined by Machniak.

Table 4
Individual developer (IND) QMM scores from extended QMM validation

Program	IND score					
	Program	QMM	Req. Mgt.	Est. Plan.	People Mgt.	Risk Mgt.
A	7	5.7	6.2	5.1	5.6	5.1
B	8	8.7	8.6	9.0	8.5	9.0
C	5	6.4	8.2	4.0	7.8	1.7
D_1	7	5.2	8.1	3.8	4.1	5.6
D_2	7.5	7.7	8.0	9.3	6.5	8.9
E	6	6.5	6.6	6.8	6.6	5.2
F	9	5.4	7.0	3.5	5.8	3.3
G	8	7.0	6.7	6.7	8.1	4.3
H	6.5	7.7	6.9	7.5	7.9	8.4
I	3	2.6	3.2	2.2	2.5	1.1
J	7	6.3	7.7	7.2	5.7	4.3

Table 5
Extended validation data correlation

	Score		
	Dev. QMM	PM success	Dev. success
PM QMM score	0.74	0.44 (0.53)	0.29
Dev QMM score		0.35 (0.49)	0.59
PM success score			0.68

In programs C, I, and J, the program manager's QMM percentage score was consistently higher than the program manager's overall program-success score and the scores of the individual development team member in the areas of overall program-success score and QMM percentage score.

Table 5 summarizes correlations of data from Tables 3 and 4. Correlations shown in parentheses are computed excluding program A where the program manager awarded a 9.9 subjective success score out of a possible 10. In general, the data shows positive correlations between subjective assessments of program-success and corresponding QMM scores, both for program managers and independent developers. This is particularly true for the data set that excludes programs A and F, as they are outliers (discussed above). In this data set, the correlation for the program manager QMM scores to program-success scores is 0.89 and the corresponding correlation for the individual development team members is 0.82. This strong correlation indicates the QMM survey instrument is producing good QMM percentage-score results.

10. Conclusion

Informal initial validation of the QMM indicated that the MM questionnaire showed a positive correla-

tion between QMM scores and observed program-success, albeit over a very limited sample set. Extended validation of the QMM showed a strong correlation with the QMM percentage score and the overall program score for both the program manager and individual development team member data sets. This indicated that the QMM survey instrument is a viable method for measuring the quality of management on a software development program for which the management policies and procedures are the same as or similar to the currently accepted software engineering management practices. In the case where the management practices and procedures are known to diverge from the currently accepted practices, the QMM survey instrument may be used to measure the level to which the currently accepted practices are implemented but will yield a lower QMM percentage score and probably predict a lower success rate for the program than the actual rate.

The QMM survey instrument may be useful in detecting discrepancies between the program manager's and the individual team member's perspectives and understanding. When the program manager's QMM percentage score is consistently higher than the overall program-success score and the QMM percentages of the individual development team members, it may indicate the program manager believes he or she is implementing and successfully using more software engineering management practices on the program than are actually being implemented by the development team.

As new metrics or ways to report metrics are discovered, both the government and the contractor have to tailor the standard metrics reporting to incorporate improvements; this ensures that the metrics set remains as useful for management purposes as possible. Adjustment of the weighting of the questions within each QMM section and among the QMM sections may be required to focus the QMM survey instrument on the areas which are the most important in determining and measuring the quality of management on a software development program. Lastly, the total number of questions might be reduced if it is determined that this is appropriate. Reducing the number of questions on the survey instrument includes examining tradeoffs related to the usefulness of the survey instrument versus the ease of executing the survey instrument if it is shorter in length.

The QMM survey instrument currently has the limitation that it does not provide specific feedback guidance for the program managers. In fact, the QMM survey instrument measures the performance of the program manager at a high level and only provides feedback to the program manager to the level of their score in each of the four QMM sections: requirements management, estimation/planning management, people management, and risk management. There is currently no provision for providing more specific feedback to the program manager other than in the area of people management, which is made up of four subsections: human resources, communication, leadership, and technical competency.

The QMM survey instrument appears to detect differences in perception between the program manager and the development team on which practices and procedures the program manager believes are implemented and working and the actual state of understanding and implementation of the practices and procedures on the development-team level. The program manager feedback mechanism could include detection of areas where the perceptions of the program manager and the development team differ greatly and the information could be used to alert the program manager or the program manager's management trainer or mentor to the problem. Knowledge of areas of potential misunderstandings enables the program manager to begin working on opening up the communication channels to better guide the development team's efforts and receive feedback on the processes and procedures which the program manager implements, enabling them to improve them.

The QMM survey instrument results could also be used as an input into current program cost, risk, and schedule estimation models to improve the resultant estimations. Currently, these models do not incorporate the quality of software development management as a factor other than to assume good management. As this assumption is not necessarily a good assumption, using the QMM results as input into the models may increase the accuracy of the estimation results.

References

[1] A.T. Bahill, F. Dean, Discovering system requirements, in: A.P. Sage, W.B. Rouse (Eds.), Handbook of Systems

Engineering and Management, Wiley, New York, 1997, pp. 175–220.
[2] B.W. Boehm, Software Engineering Economics, Prentice-Hall, Upper Saddle River, NJ, 1981.
[3] K.L. Butler, The economic benefits of software process improvement, CrossTalk—Journal of Defense Software Engineering 8 (7), 1995, pp. 14–17.
[4] A.M. Davis, D.A. Leffingwell, Making requirements management work for you, CrossTalk—Journal of Defense Software Engineering 12 (4), 1999, pp. 11–13.
[5] C. Fabian-Isaacs, E. Robinson, The project management puzzle, Software Development 7 (3), 1999, pp. S12–S16.
[6] M.A. Grossman, Validation of a quality management metric, Masters thesis, Naval Postgraduate School, Monterey, CA, September 2000.
[7] W. Hayes, J.W. Over, The personal software process: an empirical study of the impact of PSP on individual engineers, Technical report CMU/SEI-97-TR-001, Software Engineering Institute, Pittsburgh, PA, December 1997.
[8] J. Heberling, Software change management, Software Development 7 (7), 1999, pp. S7–S11.
[9] Chaos, in: The Standish Group Report, Standish Group, West Yarmouth, MA, 1995.
[10] W.S. Humphrey, A discipline for software engineering, Addison-Wesley, Reading, MA, 1995.
[11] W.S. Humphrey, Using a defined and measured personal software process, IEEE Software 13 (3), 1996, pp. 77–88.
[12] W. Keuffel, Planning for and mitigating risk, Software Development 7 (9), 1999, pp. S1–S5.
[13] J.D. Launi, Creating a project plan, Software Development 7 (5), 1999, pp. S1–S6.
[14] M.J. Machniak, Interview with Capt. (USN, Ret.) L. Preston Brooks Jr. of SAIC's Advanced Information Technology Group, 3 September 1999.
[15] M.J. Machniak, interview with Capt. (USN, ret.) Gerald Nifontoff of Lockheed-Martin Corporation's Undersea Systems Division, 7–8 September 1999.
[16] M.J. Machniak, Development of a quality management metric (QMM) measuring software program management quality, Master's thesis, Naval Postgraduate School, Monterey, CA, December 1999.
[17] M.J. Machniak, Software Program Management Focus Group No. 2, SPAWAR Systems Center, San Diego, CA, October 1999.
[18] M.J. Machniak, interview with Ms. Julie Streets of JP Training and Development Associates, 3 August 1999.
[19] M.J. Machniak, Software Program Management Focus Group No. 1, SPAWAR Systems Center, San Diego, CA, 15 October 1999.
[20] M.J. Machniak, interview with Dr. John Pickering, organizational consultant, September 1999.
[21] S. McConnell, Software's ten essentials, IEEE Software 14 (2), 1997, pp. 144.
[22] R.S. Pressman, A Manager's Guide to Software Engineering, McGraw-Hill, New York, 1993.
[23] Contracting for computer software development, FGMSD-80.4, US General Accounting Office, Washington, DC, 1979.
[24] K.E. Wiegers, Know your enemy: software risk management, Software Development 6 (10), 1998, pp. 38–42.
[25] K.E. Wiegers, Automating requirements management, Software Development 7 (7), 1999, pp. S1–S6.
[26] K.E. Wiegers, First things first: prioritizing requirements, Software Development 7 (9), 1999, pp. 48–53.

John S. Osmundson has done PhD in physics in 1968 from University of Maryland, USA. He is an associate professor of information sciences at the Naval Postgraduate School. His research interests are software project management including systematic methods for assessing the quality of project management, the systems engineering of networked information systems, and discrete-event modeling and simulation of information systems. Prior to joining NPS he was a scientist, chief systems engineer and systems engineering manager at Lockheed Missiles and Space Company.

James B. Michael obtained PhD in information technology in 1993 from George Mason University, USA. He is an associate professor of computer science at the Naval Postgraduate School. His research interests include distributed computing, information operations/warfare, and software engineering. He is a senior member of the Institute of Electrical and Electronics Engineers, and active in both the Association for Computing Machinery and the International Federation for Information Processing. He has held several research appointments, most recently with the University of California at Berkeley and Institut National de Recherche sur les Transports et leur Sécurité.

Martin J. Machniak did MS in software engineering in 1999 from Naval Postgraduate School, USA. He is the head of the Navigation and Low-Approach Landing Systems Branch at the Space and Naval Warfare Systems Center (SPAWAR), San Diego. His group provides installation, maintenance, upgrade, and development for various military and commercial systems that require en route information and precision-approach capabilities. Prior to joining SPAWAR he was a product engineer with Motorola. He currently leads a pilot program to implement Higher Performance Organization principles for improving teamwork culture and to obtain measurable improvement in areas such as customer service response time and satisfaction.

Mary Alice Grossman has done MS in software engineering in 2000 from Naval Postgraduate School, USA. She is an aerospace engineer with the NASA Dryden Flight Research Center. She is currently involved with the development of the hardware-in-the-loop flight control simulation for the C-17 Intelligent Flight Controls program. She has also worked at Edwards Air Force Base as an aerospace simulation engineer and as a program manager for the Air Warfare Mission Simulation and the Flight Simulator Modernization programs. She has extensive experience in software development and software program management.